Pixel Based Visual Mining of Geo-Spatial Data

Daniel A. Keim * Christian Panse Mike Sips

Department of Computer and Information Science, University of Konstanz, Universitätsstr. 10, Box D78, D-78457 Konstanz, Germany

Stephen C. North

AT&T Shannon Laboratory, 180 Park Avenue, Florham Park, NJ 07932-0971, USA

Abstract

In many application domains, data is collected and referenced by geo-spatial location. Spatial data mining, or the discovery of interesting patterns in such databases, is an important capability in the development of database systems. A noteworthy trend is the increasing size of data sets in common use, such as records of business transactions, environmental data and census demographics. These data sets often contain millions of records, or even far more. This situation creates new challenges in coping with scale.

For data mining of large data sets to be effective, it is also important to include humans in the data exploration process and combine their flexibility, creativity, and general knowledge with the enormous storage capacity and computational power of today's computers. *Visual data mining* applies human visual perception to the exploration of large data sets. Presenting data in an interactive, graphical form often fosters new insights, encouraging the formation and validation of new hypotheses to the end of better problem-solving and gaining deeper domain knowledge. In this paper we give a short overview of visual data mining techniques, especially for analyzing geo-spatial data. We provide examples for effective visualizations of geo-spatial data in important application areas such as consumer analysis and census demographics.

Key words: Information Visualization, Visual Data Mining, Spatial Data Mining

1 Introduction

Progress in technology allows computer systems to store and exchange datasets that were, until recently, considered extraordinarily vast. Nowadays, almost all transactions of everyday life, such as purchases made with credit cards, web pages visited, or telephone calls made are recorded by computers. This data is collected for its potential value in providing a competitive advantage to its holders. Government agencies also provide a wealth of statistical information that can be applied to problems in public health and safety, and combined with proprietary data to increase its value.

Data mining is the extraction of interesting patterns or models from observed data. Finding valuable details that reveal fine structures hidden in these already large and ever-growing data sets is difficult. With current data management systems, it is only possible to directly view very small portions of such data. With little possibility for exploring the full volume of data that was collected for its potential value, the data becomes useless and databases become 'data dumps'.

A positive trend is that visual feedback plays an increasing role in data mining. Presenting data in an interactive, graphical form often fosters new insights and encourages the formation and validation of new hypotheses, to the end of better problem-solving and deeper domain knowledge. Typically, a data analyst first specifies some parameters to restrict the search space, then runs a data mining algorithm to extract potentially interesting patterns, and examines the results graphically. For data mining to be effective, it is important to include humans in the data exploration process, combining their flexibility, creativity, and domain knowledge with the storage capacity and computational power of current computer systems. Visual data exploration thus aims at involving humans closely in data exploration, applying their perceptual abilities to the problem. Visual data mining techniques have proven to be essential in exploratory data analysis, and have high potential for the discovery of interesting patterns in very large databases.

There are many ways to approach data mining problems, including creating statistical models, clustering, and finding association rules, but in practice when data with geographic attributes are involved, it is often important to find relationships involving location. Consider, for example: credit card purchase transactions including both the address of the place of purchase and of the purchaser; telephone records including addresses or cell phone base antennae locations; space satellite remote sensed data; census and other government statistics with addresses or other geo-

* Tel.: +49 7531 88-3161; Fax.: +49 7531 88-3062
Email addresses: keim@informatik.uni-konstanz.de (Daniel A. Keim), panse@informatik.uni-konstanz.de (Christian Panse), sips@informatik.uni-konstanz.de (Mike Sips), north@research.att.com (Stephen C. North).

graphic indexes for residents; or records of property ownership based on physical locations. Often, discovering spatial patterns is crucial for understanding these data sets. Spatial data mining is the branch of data mining that deals with this problem.

2 Visual Data Mining

Visual data exploration often follows a three step process: *Overview first, zoom and filter, and then details-on-demand* (which has been called the Information Seeking Mantra (1)). In other words, in exploratory data analysis (EDA) of a data set, an analyst first obtains an overview. This may reveal potentially interesting patterns or certain subsets of the data that deserve further investigation. The analyst then focuses on one or more of these, inspecting the details of the data.

Visualization technology is essential for presenting overviews and selecting interesting subsets. For example, it is often helpful to simultaneously show the overview while focusing on subsets in a different type of visualization. An alternative is to distort the overview to focus on the interesting subsets. This can be done by allocating a larger fraction of the display to the interesting subsets in some way, while decreasing the area used for less interesting items. Good overviews of visualization techniques can be found in several recent books (2; 3; 4; 5) and a survey article (6). Similar techniques may apply to "drilling down" to inspect details of individual data items.

Visualization technology not only provides the base visualization techniques for all three steps, but also bridges the gaps between them. Visual data mining can thus be seen as a hypothesis generation process; the visualizations of the data allow the data analyst to gain insight into the data, and thereby develop and confirm new hypotheses. The verification of hypotheses may also be achieved through automatic techniques from statistics, pattern recognition, or machine learning, as a complement to visualization.

Some of they key advantages of visual data exploration over automatic data mining techniques alone are:

- yields results more quickly, with a higher degree of user satisfaction and confidence in findings
- are especially useful when little is known about the data and exploration goals are vague, because the analyst guides the search and can shift or adjust goals on the fly
- can deal with highly non-homogeneous and noisy data
- can be intuitive and require less understanding of complex mathematical or statistical algorithms or parameters
- can provide a qualitative overview of the data, allowing unexpected phenomena

to be isolated for further quantitative analysis

These factors make visual data mining indispensable in conjunction with automatic pattern extraction techniques. This is no less true for very large data sets, but more sophisticated, scalable visualization techniques may be needed. In the next sections, we show that the involvement of humans in data mining, and the application of human perceptual ability to the analysis of large data sets can provide more effective results in the mining of geo-spatial data sets.

3 Spatial Data Mining

Spatial data describes objects or phenomena with specific real-world locations. Large spatial data sets occur naturally when accumulating many samples or readings of phenomena in the real world while moving through two dimensions in space. Spatial data mining methods can be applied to understand spatial phenomena and to discover relationships between spatial and non-spatial data.

A very common approach to analyzing geo-spatial data has been to apply standard statistical analysis methods. Statistical analysis is a well understood area, and has contributed many practical algorithms. A significant problem in applying statistical methods to spatial data is that the models often assume or require statistical independence within the spatially distributed data. The difficulty is that spatial data items are often interrelated – objects are influenced by other, nearby objects. Regression models are applied to overcome this problem, but the overall analysis process is complicated.

In the development of modern data mining, researchers have proposed various automated data mining algorithms for discovering patterns in large spatial databases. These automated data mining algorithms combine methods from more mature research areas within statistics, pattern recognition and machine learning (see (7; 8) for an overview).

While automated data mining algorithms are indispensable for analyzing large geo-spatial data sets (9; 10), they often fall short of completely satisfactory results. Often automatic approaches are no better than simple visualizations of data on thematic maps. Thematic maps show spatial relationships in relation to a quantitative or qualitative data theme. Different thematic maps are examined to discover various relationships, for example, to compare median household with the general pattern of income and occupations in a geographic region.

Interactive data mining based on a synthesis of automatic and visual data mining may not only yield better results, but offer a higher degree of user satisfaction and confidence in the findings (7). Practical analysis may involve multiple parameters

shown on multiple maps. If all maps in such a collection map the data with identical coordinates, it may be possible to quickly relate parameters and to detect local correlations, dependencies, and other interesting patterns. But when large data sets are drawn on a map, identifying local patterns is greatly confounded by undesired overlap or overplotting of data points in densely populated areas, whereas lightly populated areas are nearly empty.

4 PixelMaps

This paper describes *PixelMaps*, a new way of displaying dense point sets on maps, which combines clustering and visualization. An abbreviated description of PixelMaps was published earlier (11). In brief, the idea is to first preprocess the data by kernel-density-based clustering, based on the geometric parameters (longitude, latitude) and one designated statistical parameter. This clustering is crucial to making important information visible by achieving pixel coherence with respect to the selected parameter. [1] In the next step, data points are assigned to unoccupied pixels. The assignment is done cluster by cluster, proceedings from the the densest to the least dense region of conflicting clusters. Within each region, the smallest clusters are placed first, then proceeding to larger clusters until all the points are positioned. Displayed pixels are colored according to the statistical parameter. Other statistical parameters may be shown on additional maps, with data points at the same positions as in the first map, but coloring pixels by other parameters. Such multiple linked views help to compare different statistical parameters, and may reveal local correlations, dependencies and other trends. Unfortunately a full evaluation of the density function is prohibitively expensive. This motivates work on efficient heuristic solutions.

4.1 Previous Approaches

There are several approaches to coping with dense geographic data already in common use (6). One popular method is 2.5D visualization showing data points aggregated up to map regions. This technique is in commercial systems such as VisualInsight's In3D (12) and ESRI's ArcView (13). In the example generated by In3D (figure 1(c)), we can readily see that because of aggregation, important information is lost if we are looking for patterns other than the coarsest ones. An alternative approach, showing more detail, is the visualization of individual data points as bars on a map. This technique is embodied in systems such as SGI's MineSet (14) and AT&T's Swift 3D (15). A problem here (figure 1(d)) is that too many data points

[1] Pixel coherence means similarity of adjacent pixels, which makes small pixel clusters perceivable.

are plotted at the same position, and therefore only a small portion of the data is actually displayed. Moreover, due to occlusion in 3D, some of the data is not visible unless the viewpoint is changed, so it cannot all be seen at the same time. One approach that does not aggregate the data, but avoids overlap in the two-dimensional display, is Gridfit (16). The idea is to reposition pixels locally to prevent overlap. Figure 1(b) shows an example. A problem with Gridfit is that in areas with high overlap, the repositioning depends on the ordering of the points in the database, which may be arbitrary. That is, the first data item found in the database is placed at its correct position, while subsequent overlapping data points are moved to nearby free positions, and so are locally quasi-random in their placement.

4.2 Problem Definition

The problem of visualizing geo-referenced data can be described as a mapping of input data points, with their associated original positions and statistical attributes, to unique positions on the output map. The mapping function must satisfy three main constraints. In the following, we formally define this problem. Let A be the set of input points $A = \{a_0, \ldots, a_{N-1}\}$, where $a_i = (a_i^x, a_i^y)$ is the original position of each point and $S_1(a_i), \ldots, S_k(a_i)$ are its associated statistical parameters. Because A is assumed to be large, it is likely that we have many data points i and j, for which the original positions are very close or even the same, i.e. $a_i \approx a_j$ (see figure 2(a)). Let the data display space (screen or window space) $DS \subset \mathbb{Z}^2$ be defined as $DS = \{0, \ldots, x_{max} - 1\} \times \{0, \ldots, y_{max} - 1\}$, where x_{max} and y_{max} are the maximal extension of the window. The goal of the algorithm is to determine a mapping function f of the original data set to a solution set

$$B = \{b_0, \ldots, b_{N-1}\}, \;\; 0 \leq b_i^x \leq x_{max} - 1, \;\; 0 \leq b_i^y \leq y_{max} - 1$$

such that

$$f : A \rightarrow B, \quad f(a_i) = b_i \;\; \forall i = \{0, \ldots, N-1\},$$

i.e. f determines the new position b_i of a_i. The mapping function must satisfy three constraints:

(1) **No-overlap Constraint**
The most important constraint is that all pixels are visible, which means that each must have a unique position (see figure 2(b)). Formally, this can be expressed as

$$i \neq j \;\; \Rightarrow \;\; b_i \neq b_j \quad \forall i, j \in \{1, \ldots, N-1\}$$

(2) **Position Preservation Constraint**
The second constraint is that the new positions should be ‘as close as possible” to their original ones. This can be measured by taking the absolute distance of the points from their original positions (see figure 2(c)) or as the relative distance between the data points (see figure 2(d)), leading to the following optimization goals:

- absolute position preservation

$$\sum_{i=0}^{N-1} d(a_i, b_i) \longrightarrow min$$

- relative position preservation

$$\sum_{i=0}^{N-1} \sum_{j=0, i\neq j}^{N-1} (d(b_i, b_j) - d(a_i, a_j))^2 \longrightarrow min$$

The choice between relative and absolute position preservation may depend on the application. The distance function d can be defined by an L^m-norm ($m = 1$ or 2)

$$d(b_i, b_j) = \sqrt[m]{(b_i^x - b_j^x)^m + (b_i^y - b_j^y)^m}$$

.

(3) **Clustering Constraint**

The third constraint is clustering on one of the statistical attributes $S_i, i \in \{0, \ldots, k\}$. The idea is to present the data points such that those with high similarity in S_i are positioned near each other. [2] In other words, points in the neighborhood of any given data point should have similar values, which is needed for pixel coherence (see figure 2(e)). To formalize this constraint, we need to define the neighborhood $\mathcal{N}H$ of a data point a_i, and a distance function d_S on the statistical attribute S.

$$\sum_{i=0}^{N-1} \sum_{b_j \in \mathcal{N}H(b_i)} d_S(S(b_i), S(b_j)) \longrightarrow min$$

This neighborhood function sums up all the differences on S between each data point and its neighboring points, and may be defined as

$$\mathcal{N}H(b_i) = \{b_j | d(b_i, b_j) < \varepsilon\}.$$

Because the statistical attribute may have a highly non-uniform distribution, in some cases it will be necessary to perform non-linear scaling of S before applying the distance function. Situations may also occur where there are many similar points in some areas while in others there are only a few. In this case, it may be necessary to vary ε in the area under consideration. We will return to this idea when we introduce the algorithm in section 4.4.

4.3 Trade-Offs and Complexity

While it is not hard to find a good solution for any of the three constraints taken individually, they are difficult to optimize simultaneously. Given that pixels must

[2] We assume that the clustering depends on the statistical attribute $S \in \{S_0, \ldots, S_k\}$.

not overlap as specified by constraint 1, the other two constraints often conflict. Therefore, our goal will be to find a good trade-off between constraints 2 and 3. We define a good trade-off by the following optimization problem:

$$a \cdot \sum_{i=0}^{N-1} d(a_i, b_i) + b \cdot \sum_{i=0}^{N-1} \sum_{j=0, i \neq j}^{N-1} (d(b_i, b_j) - d(a_i, a_j))^2 + c \cdot \sum_{i=0}^{N-1} \sum_{b_j \in \mathcal{N}(H(b_i))} d_S(S(b_i), S(b_j)) \longrightarrow min$$

$$a, b, c \in R$$

Unfortunately, this is a complex optimization problem and likely to be NP-hard. In general, it is difficult to find good solutions by exact techniques. In dense areas, for example, it is likely that there are data points that, although close, have different values of *S*. In a non-overlapping display, it is only possible to satisfy either constraint 2 or 3. If constraint 2 is optimized, original locations are preserved as much as possible, but there may be little pixel coherence and the visualization may not be very informative (see figure 1(b)). Conversely, if constraint 3 is optimized, the data is clustered according to *S*, but locations may be highly inaccurate.

4.4 The PixelMap Algorithm

The PixelMap algorithm solves the optimization problem by a hybrid cluster and visualization approach, based on kernel density estimation and an iterative scheme for local repositioning of data points. In this section we will describe the algorithm.

4.4.1 Basic Idea

The PixelMap algorithm starts by finding a three-dimensional kernel-density-estimation-based clustering in the dimensions $(a_i^x, a_i^y, S(a_i))$. Kernel density is a way of estimating the density of a statistical value $(S(a_i))$ at all locations in a region based on (a_i^x, a_i^y). The clustering defines sets of related pixels based on both the two spatial dimensions and the statistical parameter. The idea is to place all data points in the same cluster at proximate display pixels. The next step consists of a second kernel density estimation based clustering on the two geographical dimensions (a_i^x, a_i^y) to find dense areas. The information obtained in the two clustering steps drives the iterative positioning of data points. Starting with the densest region, all data points belonging to the same cluster are placed at neighboring pixels without overwriting previously occupied ones. If multiple clusters overlap, the smallest cluster is positioned first. After all pixels in one area are positioned, the algorithm applies the same procedure to the clusters in the next densest region, until all data points are positioned. Outliers and very small clusters, which would otherwise be treated as noise, are at last positioned at the remaining free pixels.

Procedure PixelMap Algorithm
  INPUT
    $\mathcal{D}$: Geo-Spatial Data Set
    $\mathcal{DS}$: Display Space
  OUTPUT
    PixelMap

  $KDE \longleftarrow$ DetermineKernelDenEst3D($\mathcal{D}$)
  $P = \{P_i, i = 1..n\} \longleftarrow$ DeterminePeaks(KDE)
  $C = \{C_i, i = 1..n\} \longleftarrow$ DetermineClusters(P)
  forall $C_i \in C$ **do**
    if $|C_i| < Min$ **or** Variance(C_i, Centroid(C_i)) $> \sqrt{|C_i|}$ **then**
      $C_{Noise} \longleftarrow C_{Noise} \cup C_i$
      $C \longleftarrow C \setminus C_i$
    endif
  endfor
  $Den2D \longleftarrow$ DetermineDensity2D($\mathcal{D}, C$)
  $SortC \longleftarrow$ **sort** C according to $Den2D(C_i)$ (decreasing)
  forall $cl \in SortC$ **do**
    if $|cl|$ pixels are free around $Centroid(cl)$ in $\mathcal{DS}$ **then**
      SetPixels(cl, $Centroid(cl)$, $\mathcal{DS}$)
    else
      $FreePos \longleftarrow$ FindClosestFreePixels($Centroid(cl)$, $|cl|$, $\mathcal{DS}$)
      SetPixels(cl, $FreePos$, $\mathcal{DS}$)
    endif
    forall $cl \in C_{Noise}$ **do**
      forall $p \in cl$ **do**
        if $DS[pos(p)] == empty$ **then**
          SetPixel(p, $pos(p)$, $\mathcal{DS}$)
        else
          $FreePos$ = FindClosestFreePixel(p, $pos(p)$, $\mathcal{DS}$)
          SetPixel(p, $FreePos$, $\mathcal{DS}$)
        endif
      endfor
    endfor
  endfor
EndProcedure

4.4.2 Kernel Density Estimation-based Clustering

Kernel Density Estimation (KDE) is based on the notion that the influence of each data point can be formally modeled using a mathematical function, called a kernel. For more details on KDE see other references (17; 18; 19). Typical examples of kernels are parabolic, square wave and Gaussian functions. The kernel function is

applied to each data point; an estimate of the overall density of the data space can be calculated by taking the sum of the influences of all data points. Clusters can be then mathematically determined by identifying local maxima of the overall density function, which can be found by a hill-climbing procedure. If the overall density function is continuous and differentiable at every point, the hill-climbing procedure may be guided by a standard gradient method. As an example, in the following we show several visualizations of US Year 2000 census data (*x*,*y*, median household income). Direct, static visualization of the three-dimensional density function is difficult, since the data is four dimensional. The examples of the density function shown in figures 3(a) and 3(b) result from the two dimensions longitude-income and latitude-income, respectively, both based on a Gaussian kernel. To show both geographical dimensions at once, we use a space-filling curve (figure 3(c)) to linearize the two-dimensional geographic space and plot that on the x-axis of figure 3(d). Note that the use of a space-filling curve (such as a Hilbert curve) leads to geographically clustered points. In figure 3(c), we have labeled the regions of several metropolitan areas. PixelMap allows varying the degree of clustering by changing kernel functions to adjust the definition of the neighborhood of a point. An advantage is that the placement of points in the visualization is not made individually; in fact all points within a specified neighborhood are considered. Small alterations in the data can be either smoothed away or, contrarily, emphasized, if needed by a particular application.

4.4.3 Complexity of the PixelMap Algorithm

Computing the density at a given point involves making a full database scan. Moreover, complex mathematical operations are performed. Thus computing the exact density function is time consuming, even when the database fits in main memory. To be more precise: computing the density at each point is quadratic in the number of points in the database. To speed up this process, we use a classical grid-based approximation. That is, when finding the density at *p*, only neighboring points to *p* are taken into account, because points distant from *p* do not influence the local density. So only points in the grid cell containing *p* (and, as specified by parameters, some additional neighboring cells) are accessed to calculate the density.
Because our goal is to visually cluster many points according to a statistical parameter, we must anticipate a large number ($O(n)$) of relatively small clusters. This requires computing the kernel density estimation at a fine-grained level, with many peaks that must be discovered in the hill-climbing procedure. In addition, the smoothness (σ) of the kernel function may vary with the spatial density of the input pont set, and different kernel function are needed in the spatial and statistical dimensions. These issues make it computationally prohibitive to directly implement the PixelMap algorithm for large data sets.

The next section describes an efficient heuristic algorithm to cope with the complexity of computing the three-dimensional kernel-density-estimation-based clus-

tering in $(a_i^x, a_i^y, S(a_i))$ and the two-dimensional kernel-density-estimation-based clustering in (a_i^x, a_i^y).

4.5 Fast-PixelMap Algorithm - An Efficient Implementation of the PixelMap Algorithm

In this section, we present Fast-PixelMap, an efficient algorithm that combines the advantages of gridfiles and quadtrees into a new data structure. This data structure approximates the kernel density functions to enable placement of data points at unique positions on the output map, as previously described. The combination supports recursive partitioning of Euclidian 2D space, with automatic smoothing depending on x, y density and an array-based 3D density estimation. 4.2.

4.5.1 Basic Idea

A key feature of Fast-PixelMap, as compared with the original PixelMap algorithm, is the rescaling of parts of the map to better fit dense point clouds to unique output positions. This is effective because spatial data sets in the real world are often highly non-uniformly distributed. The idea works as follows.

First, the Fast-PixelMap algorithm approximates the two-dimensional kernel-density-estimation based clustering in the two spatial dimensions (a_i^x, a_i^y) by finding a recursive partitioning of the dataset in 2D screen space, applying split operations according to the spatial parameters of the data points and the extensions of the 2D display space. The goal is (a) to find areas with high density in the two geometric dimensions (a_i^x, a_i^y) and (b) to allocate enough pixels to place all data points of these dense regions at unique positions close to each other. The top-down partitioning of the dataset and 2D screen space results in distortion of certain map regions. That means, however, virtually empty areas will shrink and dense areas will expand to achieve pixel coherence. For efficient partitioning of the dataset and the 2D screen space, and efficient scaling to new boundaries, we propose a new data structure called Fast-PixelMap. The Fast-PixelMap data structure is a combination of gridfiles and quadtrees to realize the split operations efficiently in both data and 2D display space. Our new data structure enables efficient determination of old (boundaries of the gridfile partition in the dataset) and new boundaries (boundaries of the quadtree partition in the 2D screen space) in each partition. The old and the new boundaries determine the local rescaling of various map regions. More precisely, all data points within the old boundaries will be relocated to the new positions within the new boundaries. The rescaling reduces the size of virtually empty regions and reallocates unused pixels to dense regions.

Procedure Fast-PixelMap Algorithm
INPUT

$\mathcal{D}$: Geo-Spatial Data Set
$\mathcal{DS}$: Display Space
OUTPUT
PixelMap

displaybound $\longleftarrow \{min(\mathcal{DS}.x), max(\mathcal{DS}.x), min(\mathcal{DS}.y), max(\mathcal{DS}.y)\}$
BuildFPM($\mathcal{D}$, $\mathcal{DS}$, displaybound, 0, $|D|$, 0)
L $\longleftarrow$ $\mathcal{FPM}$.leafnodes()
L $\longleftarrow$ **sort** L according to $|L_i|$ with $L_i \in L$
forall $L_i \in L$ **do**
DeterminePath(L_i)
DetermineBoundaries(0, path, $\mathcal{FPM}$)
P $\longleftarrow$ Scale2NewBoundaries($\mathcal{D}$[L_i.left:L_i.right], bound, equalbound)
PixelPlacement(P, $\mathcal{DS}$)
endfor
EndProcedure

Second, the Fast-PixelMap algorithm approximates the three-dimensional kernel-density-estimation-based clustering in the three dimensions $(a_i^x, a_i^y, S(a_i))$ performing an array based clustering for each dataset partition. After rescaling of data points to the new boundaries, the iterative positioning of data points (pixel placement step), starts with the densest regions and within the dense regions the smallest clusters are processed first. To determine the placement sequence, we sort all final gridfile partitions (leaves of the Fast-PixelMap data structure) by the number of data points contained.

A final step assigns all data points in the gridfile partition to pixels on the output map in order to provide visualizations which are as position, distance, and cluster-preserving as possible. The next sections describe the Fast-PixelMap algorithm in more detail.

4.5.2 The Fast-PixelMap Data Structure

Fast-PixelMap relies on a data structure that combines the gridfile and quadtree in a single multidimensional array, with storage for the simple array-based clustering in the third dimension.

A gridfile is a *k*-dimensional data structure for point access, that splits space into a non periodic grid. Each spatial dimension is divided by a scaling vector S_i, $i = 1 \ldots k$. The scaling vector S_i allows the definition of arbitrary split points for quadtrees. A quadtree is a degree-4 tree for storing two-dimensional data. The four children correspond to a subdivision of the 2D screen space into four quadrants (labeled by points of the compass: NE, SE, SW, NW). The coordinates of the intersections between the horizontal and vertical subdivision lines define the regions within the

split step. For geo-related 2D data sets, gridfiles and quadtrees only need two alternating split dimensions, i.e. longitude (x) and latitude (y), so we can treat both data structures as binary trees. This enables implementing both trees in a single multidimensional array where each array element represents two split operations. In other words, each array element stores the coordinates of two split points according to the pertinent split dimension x or y: (1) coordinates of the split point according to the geographical parameters (gridfile) and (2) coordinates of the split point according to the extension of the screen space (quadtree). Moreover, every array element has additional space for a constant number of classes in the third dimension. Figure 4 shows a sketch of the proposed Fast-PixelMap data structure. The advantage of this data structure is that the Fast-PixelMap algorithm can determine the old and new boundaries in time $O(log|DB|)$ (per section 4.2).

4.5.3 Fast-PixelMap Algorithm

In this section, we describe the Fast-PixelMap algorithm in detail. It has four main parts:

(1) Recursive top-down construction of the Fast-PixelMap data structure
(2) Array-based clustering
(3) Scaling to New Boundaries / Rescaling certain map regions
(4) Pixel Placement
(5) Polygon Mesh Placement

4.5.4 Recursive top-down construction

First, the Fast-PixelMap recursively builds the new data structure, top-down, using split operations. This construction starts to split on the first split dimension x. Then, in every recursive step, a partition is split into two smaller ones.

Procedure BuildFPM
INPUT
$\mathcal{D}$: Geo-Spatial Data Set
$\mathcal{DS}$: Display Space
displaybound: current boundaries within the display space
left, right: current boundaries within the data set
i: current node index within the tree
level: current split level
OUTPUT
$\mathcal{FPM}$: Fast-PixelMap Datastructure

if $level \leq MaxLevel$ or $(right - left) > 4$ **then**
if $(level \ \% \ 2) == 0$ **then**
$\mathcal{D} \longleftarrow$ **sort**$\mathcal{D}$ according to $\mathcal{D}.x$ between $[left : right]$

else
$\mathcal{D} \longleftarrow$ **sort**$\mathcal{D}$ according to $\mathcal{D}.y$ between $[left : right]$
endif
if ($level$ % 2) $==$ 0 **then**
$\mathcal{FPM}[i][0] \longleftarrow$ DetermineSplit($\mathcal{D}[left : right]$.x)
$\mathcal{FPM}[i][1] \longleftarrow (displaybound.xmin + displaybound.xmax)/2$
else
$\mathcal{FPM}[i][0] \longleftarrow$ DetermineSplit($\mathcal{D}[left : right]$.y)
$\mathcal{FPM}[i][1] \longleftarrow (displaybound.ymin + displaybound.ymax)/2$
endif
if ($level$ % 2) $==$ 0 **then**
leftbound $\longleftarrow$ displaybound
leftbound.xmax $\longleftarrow \mathcal{DS}[\mathcal{FPM}[i][1]]$
rightbound $\longleftarrow$ displaybound
rightbound.xmin $\longleftarrow \mathcal{DS}[\mathcal{FPM}[i][1]] + 1$
else
leftbound $\longleftarrow$ displaybound
leftbound.ymax $\longleftarrow \mathcal{DS}[\mathcal{FPM}[i][1]]$
rightbound $\longleftarrow$ displaybound
rightbound.ymin $\longleftarrow \mathcal{DS}[\mathcal{FPM}[i][1]] + 1$
endif
BuildFPM($\mathcal{D}$, $\mathcal{DS}$, leftbound, left, $\mathcal{D}[\mathcal{FPM}[i][0]]$, $2 \cdot i + 1$, level + 1)
BuildFPM($\mathcal{D}$, $\mathcal{DS}$, rightbound, $\mathcal{D}[\mathcal{FPM}[i][0]] + 1$, right, $2 \cdot i + 2$, level + 1)
else
forall $i \in [left : right]$ **do**
$\mathcal{FPM} \longleftarrow \mathcal{FPM}[DetermineClass(\mathcal{D}.s[i]) + 1] + 1$
endfor
$\mathcal{D} \longleftarrow$ **sort**$\mathcal{D}$ according to $|\mathcal{FPM}[2 : \sharp classes + 1]|$ between $[left : right]$
endif
EndProcedure

We apply split operations at low density positions with no more than ω (typically 10%) of $(l+r)/2$ data points. l and r are the left and the right boundaries of the gridfile partition. This avoids splitting geo-related 2D clusters, such as big cities. Figure 5 shows the impact of splitting geo-related 2D clusters to the visualization. To compute a split and to enable an efficient placement step, the Fast-PixelMap algorithm sorts the data points within the gridfile partition according to the current split dimension. The Fast-PixelMap algorithm determine the closest low density position using a histogram in the ω interval of $(l+r)/2$. In each top-down construction step, we store the coordinates of the two arising split points belonging to the current split dimension x or y. The recursion stops when the maximal split level is reached, or when the number of data points in a partition is fewer than 4 (determined experimentally). In the second step, Fast-PixelMap performs array-based clustering of all data points in the same gridfile partition according to the statistical parameter.

4.5.5 Array-based clustering

The Fast-PixelMap algorithm implements array-based clustering by partitioning the third dimension into a number of intervals starting with the minimal value. The end points of each interval are stored in an array starting with the first interval. Each interval corresponds to a class (for example, income class as defined by the US Census Bureau), and can be efficiently determined for each statistical value using binary search. Finally, the pixels are colored according to their class indices. See Figure 6 for a further illustration of the idea.

4.5.6 Scaling to New Boundaries

The third step of the Fast-PixelMap is to scale all gridfile partitions to their new boundaries in the 2D screen space. Effectively, all data points will also be scaled to new coordinates on the output map. Starting with the densest partition, the Fast-PixelMap algorithm explores the shortest path from each partition to the root element in the new data structure.

Procedure DetermineBoundaries
  INPUT
    level: current level within the data structure
    path: split history
    $\mathcal{FPM}$: Fast-PixelMap Datastructure
  OUTPUT
    bound={xmin, xmax, ymin, ymax}: current boundaries within the data set
    equalbound={xmin, xmax, ymin, ymax}: current boundaries within the
        display space

  if $level \leq MaxLevel$ **then**
    if ($level$ % 2) $==$ 0 **then**
      if $(path[level+1]$ % 2) $\neq$ 0 **then**
        bound.xmax $\longleftarrow$ $\mathcal{FPM}$[path[level]][0]
        equalbound.xmax $\longleftarrow$ $\mathcal{FPM}$[path[level]][1]
      else
        bound.xmin $\longleftarrow$ $\mathcal{FPM}$[path[level]][0]
        equalbound.xmin $\longleftarrow$ $\mathcal{FPM}$[path[level]][1]
      endif
    else
      same as x-direction, x and y exchanged
    endif
    DetermineBoundaries(path, level + 1, bound, equalbound)
  endif
EndProcedure

The shortest path is well defined in binary trees and therefore, as described above, also in the Fast-PixelMap algorithm data structure. This shortest path describes in inverse order the split history of each partition, and this split history enables determination of the old boundaries of the gridfile partition and their new boundaries in the 2D screen space. In order to determine the old and new boundaries, the Fast-PixelMap algorithm starts with the reconstruction of the split history with the first split operation. At each reconstruction step the Fast-PixelMap algorithm determines the new and old boundaries of the associated partitions, which arise through split operations, following these rules:

- If the split history runs into the left subtree (left split), set the right boundary accordingly (see Procedure DetermineBoundaries)
- If the split history runs into the right subtree (right split), set the left boundary accordingly (see Procedure DetermineBoundaries)

All splits at even levels are split operations in the x dimension, and the rest are split operations in the y dimension. We store the old boundaries in an array called *boundy* and the new boundaries in an array called *equalbound* and use the formula

$$new = equalbound.min + (old - bound.min) \cdot \frac{(equalbound.max - equalbound.min)}{bound.max - bound.min}$$

to scale all data points to their new coordinates.

4.5.7 Pixel Placement Step

The last step of the Fast-PixelMap algorithm is the iterative placing of all data points at unique positions on the output map, in a way that yields informative visualizations. Note, that the rescaling of gridfile partitions on the 2D screen space solves neither the pixel overlap nor the pixel coherence problem. The basic idea of the pixel placement algorithm is the iterative positioning of all data points, depending on the class they belong to. More precisely, the pixel placement algorithm divides all classes into two partitions: cluster and non-cluster. To solve the pixel coherence problem, real clusters in $(a_i^x, a_i^y, S(a_i))$ are placed first. The pixel placement algorithm computes the variance of all classes to identify real clusters in the three dimensions $(a_i^x, a_i^y, S(a_i))$. A class is considered a real cluster if its variance is less than some constant *MaxVariance*.

Procedure Pixel Placement
INPUT
P: data points belonging the same partition P
$\mathcal{DS}$: Display Space
OUTPUT
PixelMap

forall $P_i \in P$ **do**
if $|P_i| < Min$ **or** Variance(P_i, Centroid(P_i)) $> \sqrt{|P_i|}$ **then**
$C_{Noise} \longleftarrow C_{Noise} \cup P_i$
else
$C \longleftarrow C \cup P_i$
endif
endfor
C $\longleftarrow$ **sort** C according $|P_i|$ with $P_i \in C$
forall $C_i \in C$ **do**
if $|C_i|$ pixels are free around $Centroid(C_i)$ in $\mathcal{DS}$ **then**
SetPixels(C_i,$Centroid(C_i)$)
else
$FreePos$ = FindClosestFreePixels(C_i, $Centroid(C_i)$, $\mathcal{DS}$)
SetPixels(C_i, $FreePos$, $\mathcal{DS}$)
endif
endfor
forall $C_i \in C_{Noise}$ **do**
if $\mathcal{DS}[pos(p)] == empty$ **then**
SetPixel(p, $pos(p)$, $\mathcal{DS}$)
else
$FreePos$ = FindClosestFreePixel(p, $pos(p)$, $\mathcal{DS}$)
SetPixel(p, $FreePos$, $\mathcal{DS}$)
endif
endfor
EndProcedure

Next, the placement algorithm tries to place all cluster points at free positions close to the cluster's centroid, without overwriting already-occupied pixels. This second step is performed starting with the smallest class from the real cluster partition. Small clusters take the fewest free positions and in practice can often be placed optimally. To solve the pixel coherence problem and make small clusters visible, all cluster members are placed close to their centroid. If data points cannot be placed without overwriting existing pixels, the placement algorithm searches for the next closest free region to the centroid, in which most of the data points can be placed. For all other data points, the placement algorithm searches for the closest free pixel for each. Finally, the placement algorithm continues with the smallest class from the non-cluster partition to place all data points at free pixels without overwriting the occupied ones. If a data point cannot be placed, the placement algorithm searches for the closest free pixel in the solution set. Figure 8 shows the idea of the placement step and figure 9 displays an example of the described placement in the Manhattan area. The bipolar colormap encodes the income classes. Blue color is used for low income and red for high income. The lightness of the color corresponds to the number of class members. The color usage described above can be seen in figure 9(a).

4.5.8 Polygon Mesh Placement Step

Often, a map has an associated polygonal mesh of geo-political boundaries that help in identifying locations. Assuming this mesh is given along with the input points *A*, we would like to provide a transformed mesh for the output map. The vertices of the mesh are handled in a way similar to data points. Each vertex is repositioned separately: first the cell of the quadtree containing the vertex is found. Then, the new position of the vertex is calculated by scaling the cells of the quadtree, the original boundaries in the data set, to the new boundaries in 2D display space. To calculate the new position of each vertex, the same algorithm as described in 4.5.6 is used. By repositioning each vertex, we iteratively construct the transformed polygon mesh. For further details on polygon mesh and distortion techniques see previous work (20).

4.5.9 Complexity

The time complexity of the proposed approach is $O(nlog^2n)$ and the additional space overhead, $0(n+logn)$, is negligible. This additional space is needed by the Fast-PixelMap data structure to store the original data points and a constant number of split-operations (depending on the maximal split-level).

5 Application and Evaluation

For comparison, we implemented alternative methods of generating PixelMaps that likewise attempt to optimize the goals presented in section 4.2. We compared the Fast-PixelMap algorithm with a genetic multi-objective optimization algorithm (21; 22) and a PixelMap algorithm based on the DenClue clustering algorithm (23) using both absolute and relative position preservation as defined in constraints 1 and 2 and clustering effectiveness defined in constraint 3. Experiments are conducted using a sample of 30000 points from the United States Year 2000 Census Median Household Income database (24).

5.1 Efficiency Evaluation

The average number of data points assigned to the same position has a heavy influence on the performance of the Fast-PixelMap algorithm. To evaluate its efficiency, we measured computation time based on the sample of 30000 points, with varying degrees of overlap (number of pixels assigned to the same position). Figure 10 shows, for differing screen resolutions, the corresponding degree of overlap. Figure 11 shows time-performance curves of the genetic algorithm, Fast-PixelMap,

and DenClue-PixelMap under varying degrees of overlap. We ran the experiments on a 700 MHz Intel Pentium-3 computer with 1 GB of main memory. We can easily see that the computation time of all three methods increases exponentially with the degree of overlap, and for high degrees of overlap it is nearly impossible to find solutions for the criteria defined in section 4.2. The results also indicate that the Fast-PixelMap algorithm outperforms the other two methods for all degrees of overlap, and is efficient enough to be practical on large spatial data sets.

5.2 Effectiveness Evaluation

More important in visual data mining than the efficiency of computing visualizations is their effectiveness. This can be evaluated by visual inspection and comparison of the generated displays. It can also be mathematically measured in relation to the three optimization goals defined in section 4.2. Before presenting the visual evaluation and applications. we first define the measurement of absolute and relative position preservation errors and clustering error. The effectiveness was measured for the same sample of 30000 points from the U.S. Year 2000 Census Median Household Income database mentioned previously. The measured curves for the absolute and relative position preservation and clustering degree are shown in figure 12. The following facts can be observed.

First, figure 12 shows that the genetic multi-objective optimization algorithm provides the smallest average deviation from the original position, and so, the smallest absolute position preservation error. The results show that both PixelMap algorithms (Fast-PixelMap and DenClue-PixelMap) provide similar average deviation from the original positions. The measured error for Fast-PixelMap is slightly higher. The higher absolute position preservation error can be explained by the local rescaling of certain parts of the map. On the other hand, the relative position preservation error is much lower than for the two other methods. The lower relative position preservation error can be explained by the fact that dense regions and virtually empty regions expand or shrink respectively, which allows placing all data points at pixel positions close to the desired ones. Note that for all three methods, the absolute position preservation constraint becomes is exponentially costly to satisfy for higher degrees of overlap. This follows from the fact that the number of data points assigned to the same pixels increases with the degree of overlap. That means, however, that the number of available pixels in dense regions rapidly decreases, and the distance to the next free available pixels grows in an exponential fashion.

Second, the Fast-PixelMap algorithm provides the smallest average deviation of the relative position of the data points from each other in comparison with the other two methods. An interesting result is that the average deviation of the relative position of the data points to each other of the genetic algorithm fluctuates at high error

levels. This can be explained by the fact that DenClue-PixelMap and the genetic algorithm place all data points at positions on a traditionally designed dot map (without distortion). Consequently, in dense regions, only a small portion of the data points can be placed close to the desired positions. For the rest of the data points, both algorithms search for the next free pixel. The distance to the next free available pixels grows exponentially in medium and high degrees of overlap.

Finally, figure 12 shows that DenClue-PixelMap provides the best clustering. Fast-PixelMap and the genetic algorithm provide similar clustering results. It is clear that, on the one hand, the DenClue kernel-density-estimation based clustering finds real clusters in the three dimensions $(a_i^x, a_i^y, S(a_i))$, and on the other hand, the other two methods are not real cluster algorithms for 3D point clouds. However, the result for the Fast-PixelMap algorithm is acceptable for real time applications. The same can be observed for the genetic multi-objective optimization approach.

Concluding, we observe that the Fast-PixelMap algorithm provides good approximations to the kernel-density-estimation based clustering PixelMap algorithm. The advantage of the Fast-PixelMap algorithm is that it computes PixelMaps in real-time for low and medium degrees of overlap, and after some seconds for high degrees of overlap. The measured optimization results are comparable to the other two methods. Finally, we can observe that for very high degrees of overlap, it is impossible to find assignments with small position and clustering errors, and the measured error functions become even more exponential under all three methods.

5.3 Visual Evaluation and Applications

Formal effectiveness measures, such as the absolute and relative position preservation and clustering errors considered above, are of limited value if they do not correspond to useful visualizations. In this section, we provide a visual comparison of the PixelMap technique with traditional approaches, which in general confirm the measured mathematical criteria.

Our first comparison (see Figure 13) is a map of the United States showing the U.S. Year 2000 Median Household Income Data. The left visualization is a traditional map, and the right visualization was made by the Fast-PixelMap algorithm. The traditional map provides random results in areas with high degree of overlap while sparsely populated areas are virtually empty. The visualization on the right shows the advantages of the Fast-PixelMap algorithm. First, we can easily see that New York City and Los Angeles County are the population areas of greatest interest in the United States. In the Fast-Pixelmap visualization, the densest regions get the space needed to place all data points close to each other. We can even see the distribution of U.S. Year 2000 Median Household Incomes in these regions. Finally, for the U.S. Year 2000 Median Household Income we can observe a sharp decline

between high and low income. Most Americans had income below $75000 U.S., and ca. 10% of all Americans live below the federal poverty level.

Our second comparison (see Figure 14) is a zoomed view to an interesting region, the State of New York. We can easy see that households with very high median household income are located in Manhattan and Queens, and households with low median household income are in the Bronx and Brooklyn. Especially, very wealthy inhabitants live on the east side of Central Park. The degree of overlap for a screen resolution of 1200x1200 is 0.66 for the whole United States and 0.82 for the State of New York.

6 Conclusions

In this paper, we presented PixelMap, a novel pixel-based visual data mining technique that combines kernel-density-based clustering with visualization, and gave an efficient approximation for displaying large amounts of spatially referenced data. PixelMap avoids the problem of losing information because of overplotting data points. More precisely, it assigns each data point to a unique pixel in the 2D display space, and tries to achieve a good trade-off between spatial locality (absolute and relative position preservation) and clustering to solve the pixel-coherence problem. We applied a number of real data sets to evaluate the Fast-PixelMap algorithm. The proposed Fast-PixelMap algorithm provides an effective, efficient solution to the optimization problem defined in this paper, and is of practical value for exploring spatially referenced statistical data. In future work, we expect to investigate related approaches for visualizing large geographical data sets. One idea is to combine the pixel placement technique with a cartogram algorithm, which first computes a distortion of the output map having low shape and area error, and then places pixels on this map with a simple array-based clustering.

7 Acknowledgments

We thank Carmen Sanz Merino and Hartmut Ziegler for their great support. We thank Dave Belanger and Mike Wish for encouraging this investigation.

References

[1] B. Shneiderman, The eyes have it: A task by data type taxonomy for information visualizations, in: Proc. IEEE Visual Languages, 1996, pp. 336–343.

[2] S. Card, J. Mackinlay, B. Shneiderman, Readings in Information Visualization, Morgan Kaufmann Publishers, 1999.
[3] H. Schumann, W. Müller, Visualisierung: Grundlagen und allgemeine Methoden, Springer, 2000.
[4] B. Spence, Information Visualization, Pearson Education Higher Education Publishers, UK, 2000.
[5] C. Ware, Information Visualization: Perception for Design, Morgan Kaufmann Publishers, 2000.
[6] G. Geisler, Making information more accessible: A survey of information visualization applications and techniques, http://www.ils.unc.edu/~geisg/info/infovis/paper.html, Feb. 2003.
[7] J. Han, M. Kamber, Data Mining: Concepts and Techniques, Morgan Kaufmann Publishers, 2001.
[8] D. J. Hand, H. Mannila, P. Smyth, Principles of Data Mining, MIT Press, 2001.
[9] K. Koperski, J. Adhikary, J. Han, Spatial data mining: Progress and challenges, in: Research Issues on Data Mining and Knowledge Discovery, 1996.
[10] K. Koperski, J. Han, J. Adhikary, Mining knowledge in geographical data, Comm. A.C.M. .
[11] D. A. Keim, S. C. North, C. Panse, M. Sips, Pixelmaps: A new visual data mining approach for analyzing large spatial data sets, in: ICDM 2003, The Third IEEE International Conference on Data Mining, Melbourne, Florida, USA, 2003.
[12] Advizor Solutions, Inc., Visual Insight In3D, http://www.advizorsolutions.com/, Feb. 2003.
[13] ESRI, Arc View, http://www.esri.com/software/arcgis/arcview/index.html, Feb. 2003.
[14] SGI MineSet, website, http://www.sgi.com/software/mineset.html, Feb. 2002.
[15] D. Keim, E. Koutsofios, S. C. North, Visual exploration of large telecommunication data sets, in: Proc. Workshop on User Interfaces In Data Intensive Systems (Invited Talk), Edinburgh, UK, 1999, pp. 12–20.
[16] D. A. Keim, A. Herrmann, The gridfit algorithm: An efficient and effective approach to visualizing large amounts of spatial data, Proc. IEEE Visualization, Research Triangle Park, NC (1998) 181–188.
[17] B. W. Silverman, Density Estimation for Statistics and Data Analysis, Chapman & Hall, 1986.
[18] D. W. Scott, Multivariate Density Estimation, Wiley and Sons, 1998.
[19] M. P. Wand, M. C. Jones, Kernel Smoothing, Chapman & Hall, 1995.
[20] D. A. Keim, S. C. North, C. Panse, J. Schneidewind, Efficient cartogram generation: A comparison, in: Proc. InfoVis 2002, IEEE Symposium on Information Visualization, Boston, Massachusetts, 2002, pp. 33–36.
[21] E. Zitzler, Evolutionary algorithms for multiobjective optimization: Methods and applications, Ph.D. thesis, Swiss Federal Institute of Technology (ETH) Zurich, tIK-Schriftenreihe Nr. 30, Diss ETH No. 13398, ISBN 3-8265-6831-

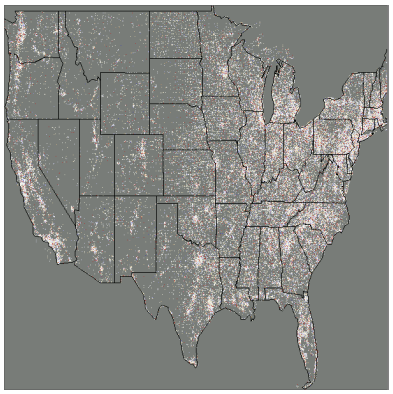
1, http://www.tik.ee.ethz.ch/~zitzler (1999).
[22] E. Zitzler, L. Thiele, Multiobjective optimization using evolutionary algorithms - a comparative case study. parallel problem solving from nature, in: PPSN-V, 1998, pp. 292–301.
[23] A. Hinneburg, D. A. Keim, An efficient approach to clustering in large multimedia databases with noise, in: Knowledge Discovery and Data Mining, 1998, pp. 58–65.
[24] United States Department of Commerce, Census Bureau website, http://www.census.gov/, Sep. 2003.

Daniel A. Keim is a full professor and head of the database and visualization group in the Department of Computer and Information Science of the University of Konstanz, Germany. His research interests include data mining and information visualization, as well as similarity search and indexing in multi-media databases. He is a member of the IEEE CS, ACM, and GI (the German Society for Informatics).

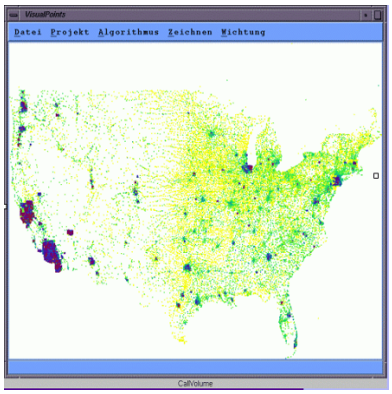
Stephen C. North is department director of Information Visualization Research in the AT&T Shannon Laboratory in Florham Park, New Jersey, U.S.A. His research interests include graph layout and visualization for spatial data mining. He is a member of the ACM and a Senior Member of IEEE.

Christian Panse is a PhD student in the Department of Computer and Information Science of the University of Konstanz, Germany. His research interests include data mining and information visualization. He is a member of the IEEE CS.

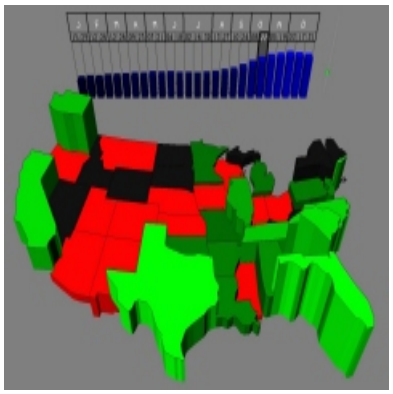
Mike Sips is a PhD student in the Department of Computer and Information Science of the University of Konstanz, Germany. His research interests include data mining and information visualization. He is a member of the IEEE CS, ACM, and GI (the German Society for Informatics).



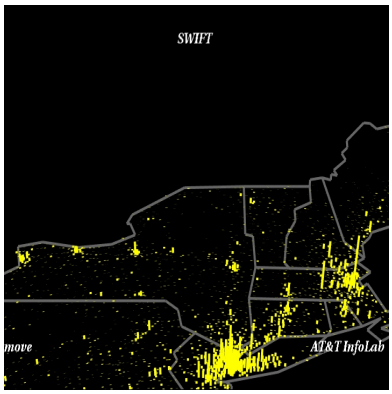
(a) Traditional 2D Map (with overlap)



(b) Non-overlap 2D Map (Gridfit)

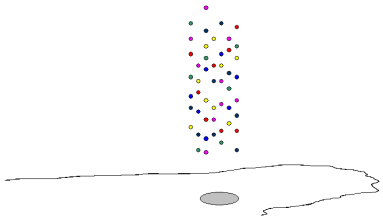


(c) 2.5D Aggregated Map (In3D)

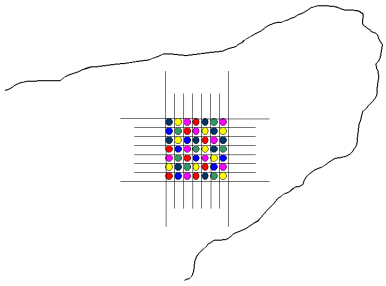


(d) 2.5D Bar Map (Swift)

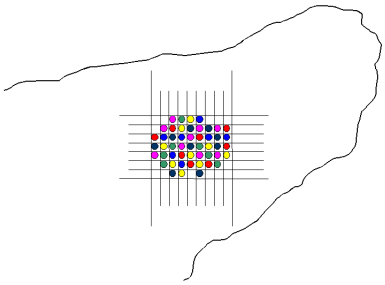
Fig. 1. *Approaches to Visualize Large Spatial Data Sets* - An Overview



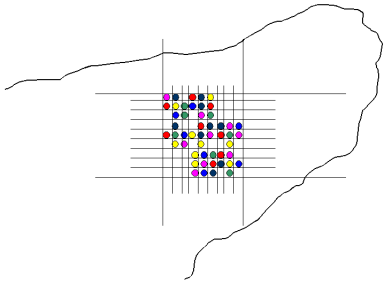
(a) Dense data points in 3D space (x,y,s)



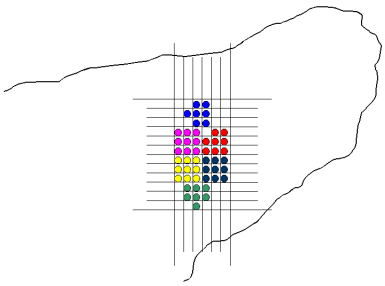
(b) No-Overlap Constraint



(c) Absolute Position Preservation

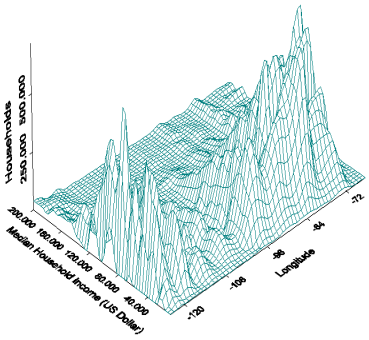


(d) Relative Position Preservation

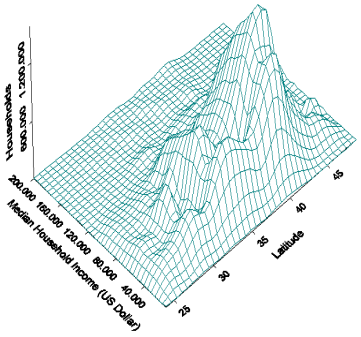


(e) Clustering Constraint

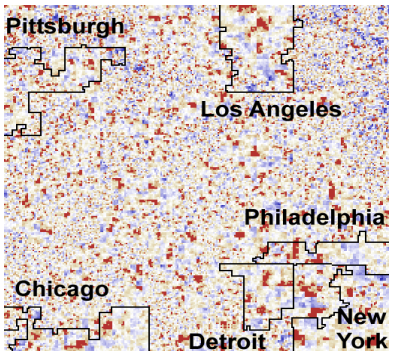
Fig. 2. *Problem Definition Constraints* - The goal is to find a good trade-off between constraints 2(c), 2(d) and 2(e), such that constraint 2(b) is always satisfied



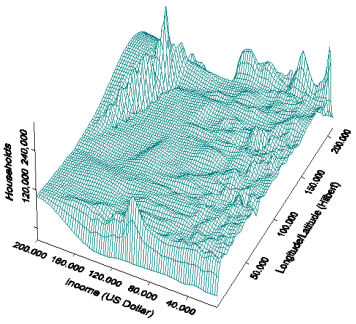
(a) *2D Density Plot (longitude, median household income)* - The two densest median household income areas (Atlantic Coast and Pacific Coast) regions have up to $100.000 U.S. median household income; the two lowest density regions are the New England and Rocky Mountain regions



(b) *2D Density Plot (latitude, median household income)* - Only significant median household income for the United States middle latitude region



(c) *Space-filling (Hilbert) Curve* - Yields a good clustering of median household income with respect to six cities that are identified, but the geographical relationship of the clusters is lost



(d) *Linearized 2D Density Plot (xy, income) of the Space-filling (Hilbert) Curve* - Very wealthy people ($200.000 U.S. and above) yield significant geo-related clusters

Fig. 3. *2D Visualizations of the 3D Density Function* - United States Year 1999 Median Household Income

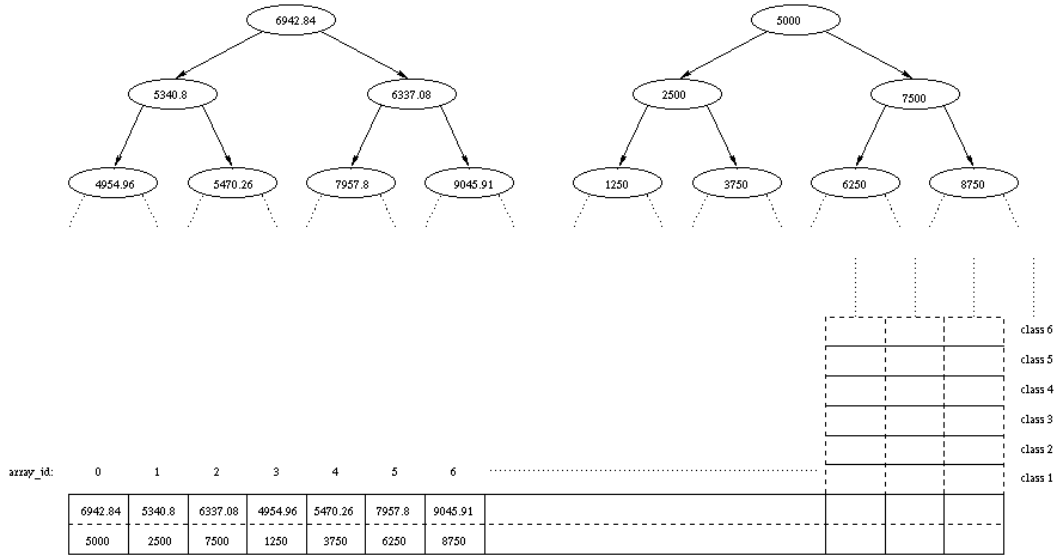
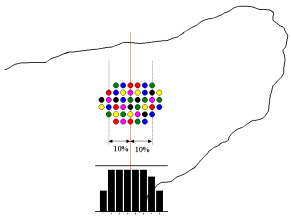
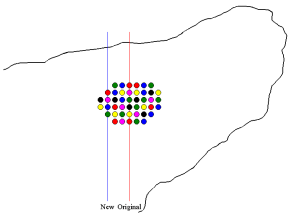


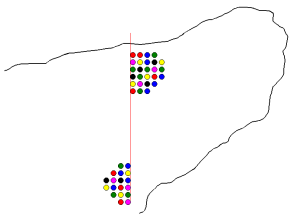
Fig. 4. *The Fast-PixelMap Data Structure: Combination of Gridfile and Quadtree* - Gridfile and quadtree represented as binary trees in a single multidimensional array in which each array element represents two split operations



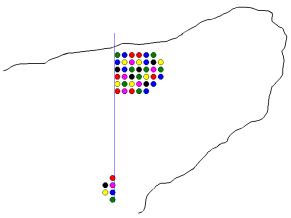
(a) Split operations at low density positions with ω (typically 10%) of $(l + r)/2$



(b) Two different split operations - Split operation at the center of a partion and split operation at low density position

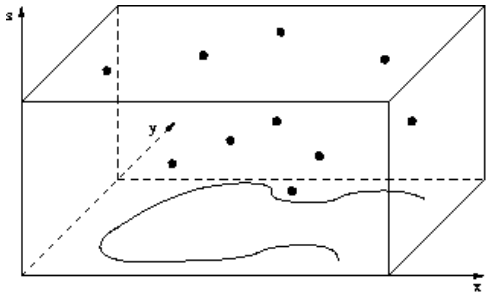


(c) Unwanted splitting geo-related 2D clusters (split operation at the center of a partion)

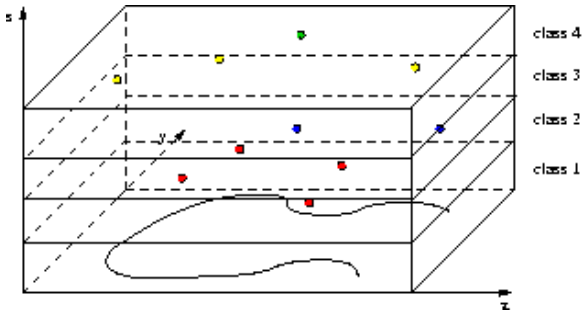


(d) Avoidance of splitting geo-related 2D clusters (split operation at low density position)

Fig. 5. Splitting geo-related 2D clusters

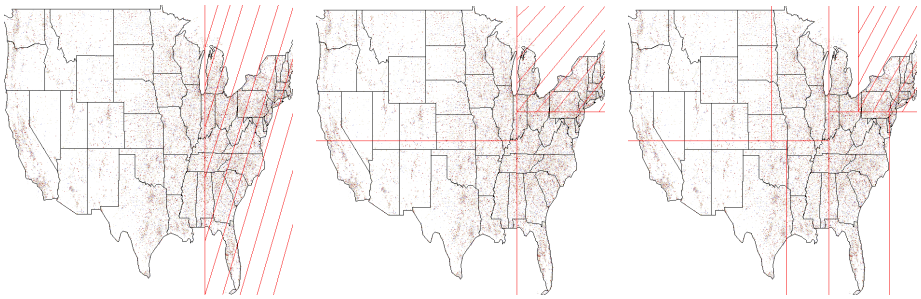


(a) Dataset partition and its data points



(b) Array-based clustering

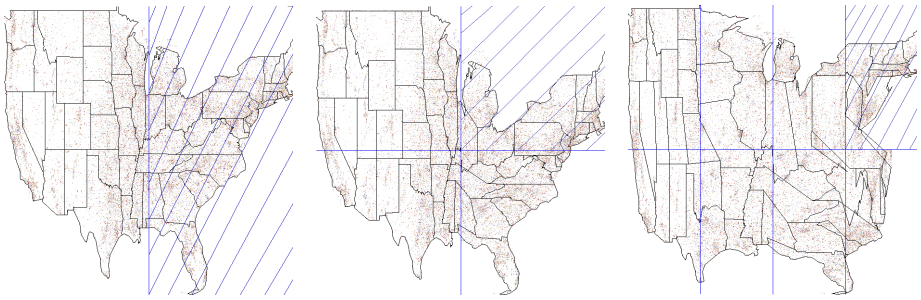
Fig. 6. Array-based clustering in the three dimensions $(a_i^x, a_i^y, S(a_i))$



(a) Original boundaries after first split

(b) Original boundaries after second split

(c) Original boundaries after third split

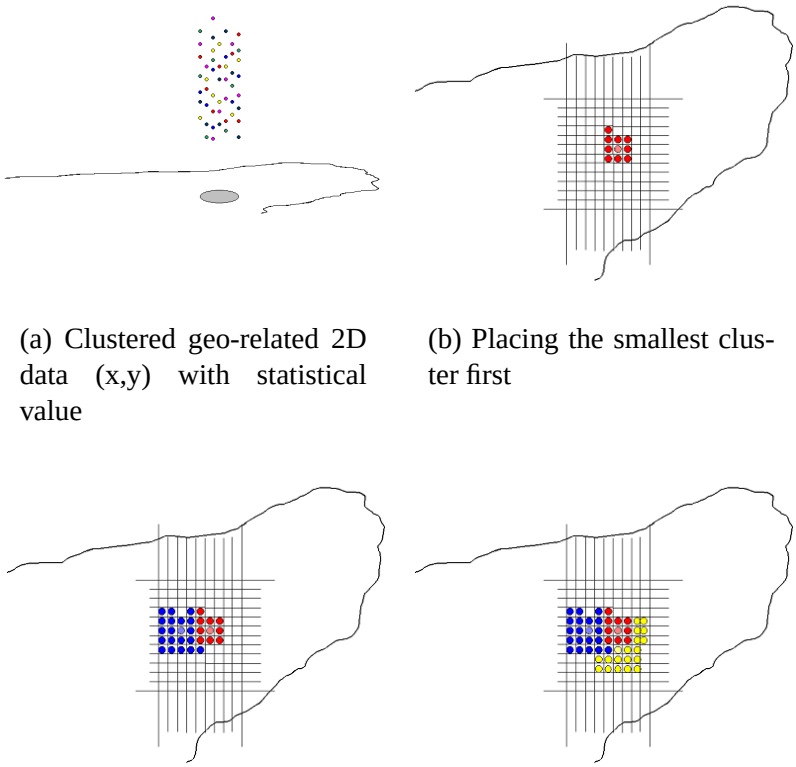


(d) New boundaries after first split

(e) New boundaries after second split

(f) New boundaries after third split in the 2D screen space

Fig. 7. *Scale to new boundaries* - scale all gridfile partitions to their new quadtree boundaries in the 2D screen space



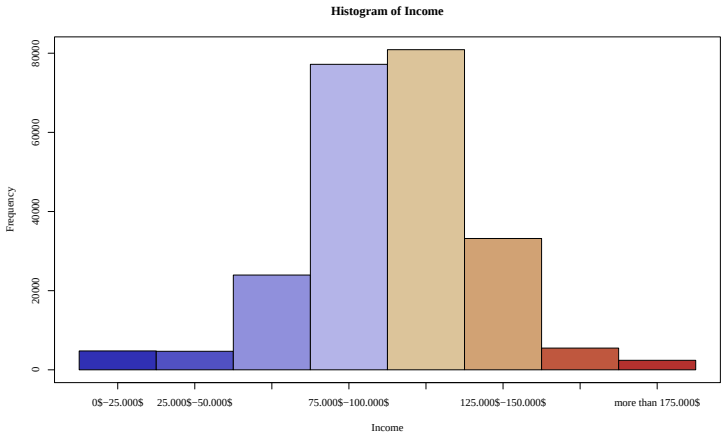
(a) Clustered geo-related 2D data (x,y) with statistical value

(b) Placing the smallest cluster first

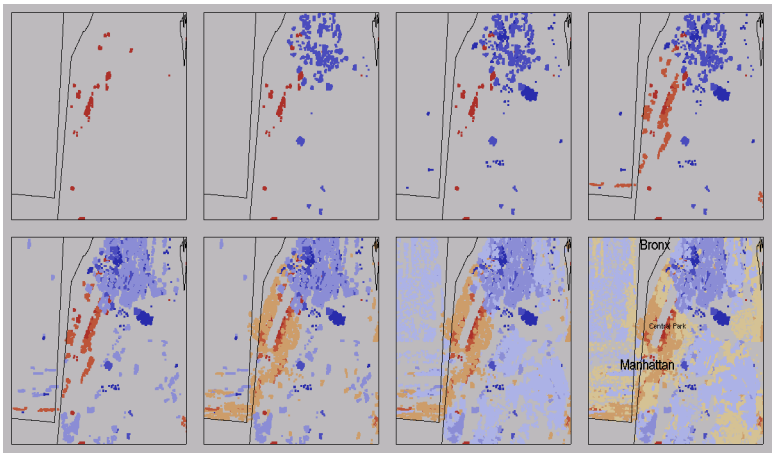
(c) Placing the next cluster in the closest free screen region

(d) Placing the next cluster in the same way

Fig. 8. *Pixel placement step* - starting with the smallest class from the real cluster partition and placing all class members at free positions close to their centroid without overwriting existing pixels



(a) The plot displays the histogram of the eight median household income classes. Income classes are superposed in color and correspond to the colors in figure 9(b)



(b) Pixel Placement Step starting with the smallest cluster (left to right)

Fig. 9. *Pixel placement step for real 3D Clusters in the Manhattan Area* - starting with the smallest class from the real cluster partition and placing all cluster members to free positions close to their centroid without overwriting pixels

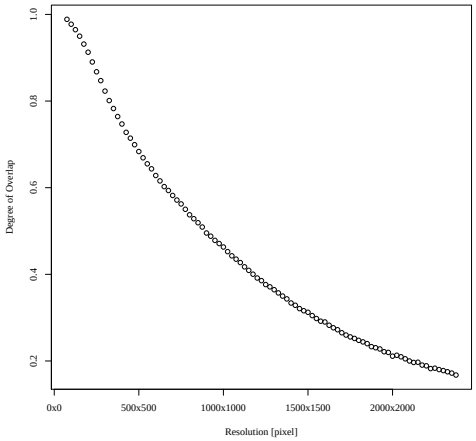


Fig. 10. *Varying degree of pixel overlap depending on screen resolution* - even for a modern screen resolution of 1600*x*1200 the degree of overlap is about 0.4; 40% of our sample of data points (about 12000 points) from the U.S. Year 2000 Census Median Household Income database are cannot be directly placed without overwriting already-occupied pixels.

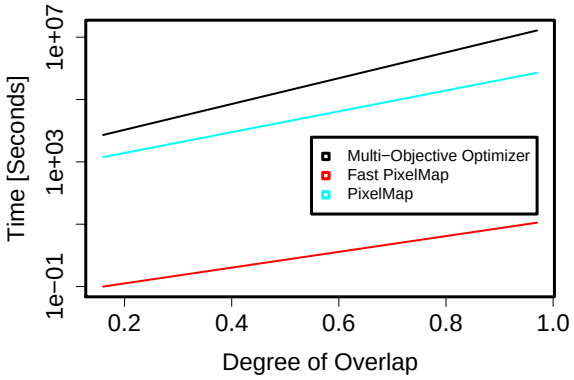
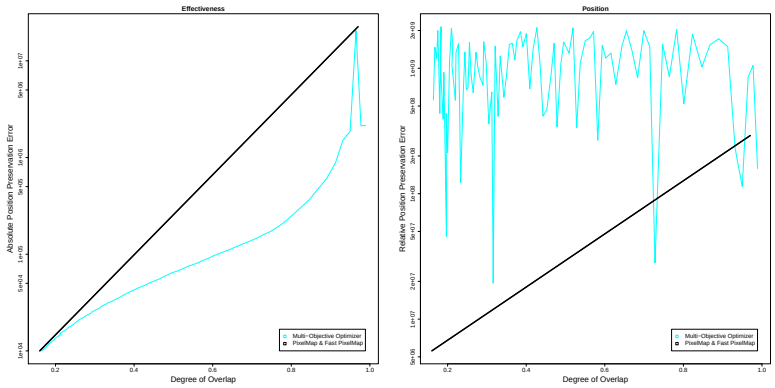
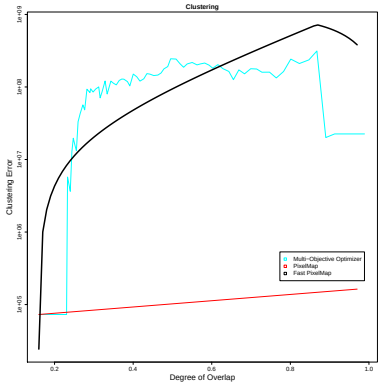


Fig. 11. *Efficiency of the (a) genetic algorithm for multi-objective optimization, (b) Fast-PixelMap, and (c) PixelMap algorithm based on the DenClue clustering algorithm to find solutions for the in section 4.2) defined optimization problem with a increasing degree of overlap, logarithmic scale* - It becomes exponentially difficult to compute PixelMaps which optimizes the defined optimization goals, that means, however, to find good trade-off between the defined optimization constraints in section 4.2, the Fast-PixelMap algorithm outperforms very clear the other two methods for medium and high degrees of overlap



(a) (b)



(c)

Fig. 12. *Effectiveness Measurement of the defined optimization constraints 1, 2, and 3 in section 4.2* - We can observe, that the Fast-PixelMap algorithm provides a good approximation of the kernel-density-estimation based clustering PixelMap algorithm. The advantage of the Fast-PixelMap algorithm is that it computes PixelMaps in real-time (low and medium degrees of overlap) and after some seconds (high degrees of overlap). The Fast-PixelMap algorithm provides an effective and efficient solution to the optimization problem defined in section 4.2, and is of practical value for exploring spatially referenced statistical data. The measured constraint errors defined in section 4.2 are comparable to the DenClue-PixelMap and to the genetic algorithm for multi-objective optimization.

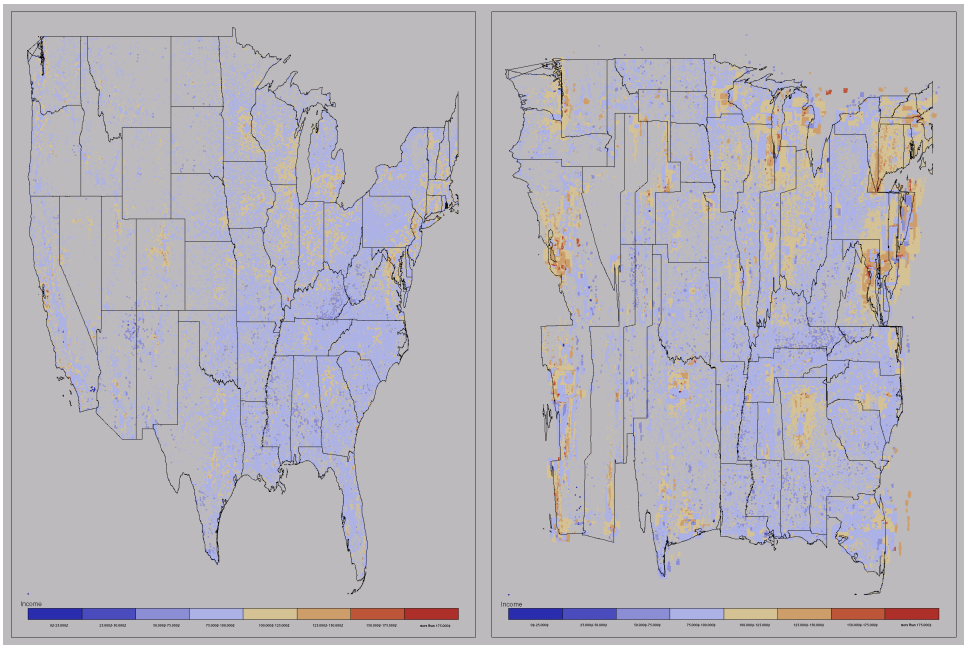


Fig. 13. *Comparison of Traditional Map versus PixelMap* – United States, Year 1999 Median Household Income.

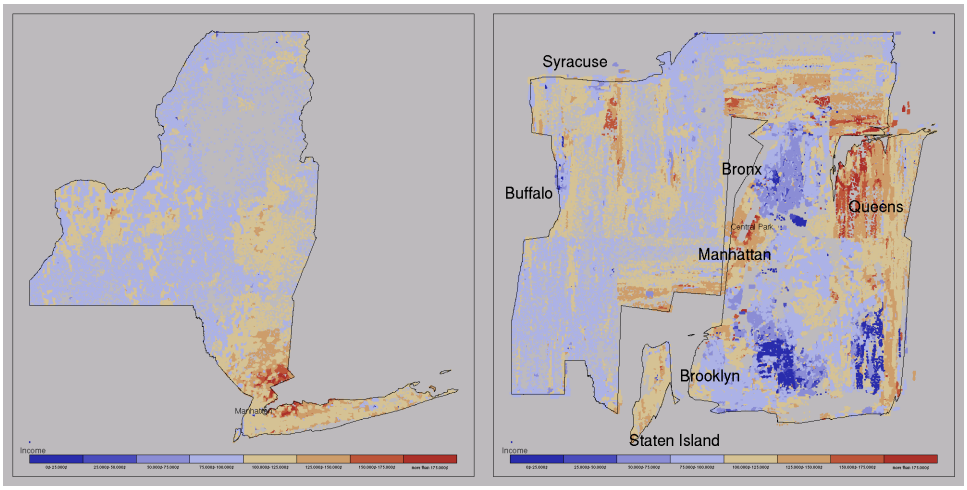


Fig. 14. *Comparison of Traditional Map versus PixelMap* - New York State, Year 1999 Median Household Income. This map displays cluster regions e.g. on the East side of Central Park in Manhattan, where inhabitants with high income live, or on the right side of Brooklyn, where inhabitants with low income live.